45장 프로미스

> 💡 **배경**
> 자바스크립트는 비동기 처리를 위한 하나의 패턴으로 콜백 함수를 사용한다.
> 하지만 전통적인 콜백 패턴은 콜백 헬로 인해 가독성이 나쁘고,
> 비동기 처리 중 발생한 에러의 처치가 곤란하며 ,
> 여러 개의 비동기 처리를 한번에 처리하는 데도 한계가 있어
> 프로미스를 도입하였다.

45.1 비동기 처리를 위한 콜백 패턴의 단점

45.1.1 콜백 헬

- 비동기 함수를 호출하면 함수 내부의 비동기로 동작하는 코드가 완료되지 않았다 해도 기다리지 않고 즉시 종료

  ⇒ 비동기 함수 내부의 비동기로 동작하는 코드는 비동기 함수가 종료된 이후에 완료

  ⇒ 비동기 함수 내부의 비동기로 동작하는 코드에서 처리 결과를 외부로 반환하거나 상위 스코프의 변수에 할당하기는 어려움

```
let g = 0

// 비동기 함수인 setTimeout 함수는 콜백 함수의 처리 결과를 외부로 반환하거나
// 상위 스코프의 변수에 할당하지 못한다.
setTimeout(() => { g = 100; }, 0);
console.log(g) // 0
```

- **콜백 헬**

콜백 함수를 통해 비동기 처리 결과에 대한 후속 처리를 수행하는 비동기 함수가

비동기 처리결과를 가지고 또 다시 비동기 함수를 호출하여 복잡해지는 현상

```
get('/step1', a => {
    get('/step2/${a}', b => {
        get('/step3/${b}', c => {
            get('/step4/${a}', d => {
                console.log(d);
            });
        });
    });
});
```

45.1.2 에러 처리의 한계

비동기 처리를 위한 콜백 패턴의 문제점 중에서 가장 심각한 것은 에러 처리가 곤란하다는 것

```
try {
  setTimeout(() => { throw new Error('Error!'); }, 1000);
} catch (e) {
  // 에러를 캐치하지 못한다.
  console.log('캐치한 에러', e);
}
```

45.2 프로미스의 생성

- **Promise**는 ECMAScript 사양에 정의된 표준 빌트인 객체
- **Promise** 생성자 함수는 비동기 처리를 수행할 콜백함수를 인수로 전달받는데

  이 콜백함수는 resolve와 reject 함수를 인수로 전달받는다.

```
// 프로미스 생성
const promise = new Promise((resolve, reject) => {
  // Promise 함수의 콜백 함수 내부에서 비동기 처리를 수행한다.
  if( /*비동기 처리 성공*/ ){
    resolve('result');
  } else { /* 비동기 처리 실패 */
    reject('failure reason');
```

```
  }
});
```

비동기 처리가 성공하면 콜백 함수의 인수로 전달받은 resolve 함수를 호출하고,

비동기 처리가 실패하면 reject 함수를 호출한다.

비동기 작업이 가질 수 있는 3가지 상태

프로미스의 상태 정보	의미	상태 변경 조건
pending(대기 상태)	비동기 처리가 아직 수행되지 않은 상태	프로미스가 생성된 직후 기본 상태
fulfilled(성공)	비동기 처리가 수행된 상태(성공)	resolve 함수 호출
rejected(실패)	비동기 처리가 수행된 상태(실패)	reject 함수 호출

- 비동기 처리 성공: resolve 함수를 호출해 프로미스를 fulfilled 상태로 변경
- 비동기 처리 실패: reject 함수를 호출해 프로미스를 rejected 상태로 변경

45.3 프로미스의 후속 처리 메서드

프로미스의 비동기 처리 상태가 **변화**하면 후속 처리 메서드에 인수로 전달한 콜백 함수가 선택적으로 호출

⇒ 프로미스의 처리 결과가 인수로 전달

모든 후속 처리 메서드는 프로미스를 반환하며 비동기로 동작

45.3.1 Promise.prototype.then

`then` 메서드는 두 개의 콜백 함수를 인수로 전달받는다.

- 첫 번째 콜백 함수: 프로미스가 fulfilled 상태가 되면 호출

  프로미스의 비동기 처리 결과를 인수로 전달받는다.
- 두 번째 콜백 함수: 프로미스가 rejected 상태가 되면 호출

  프로미스의 에러를 인수로 전달받는다.

💡 then메서드는 언제나 프로미스를 반환

45.3.2 Promise.prototype.catch

`catch` 메서드의 콜백함수는 프로미스가 rejected상태인 경우만 호출

45.3.3 Promise.prototype.finally

finally 메서드의 콜백 함수는 프로미스의 성공 또는 실패와 상관없이 무조건 한 번 호출

45.4 프로미스의 에러 처리

- catch 메서드를 모든 then 메서드를 호출한 이후에 호출하면 비동기 처리에서 발생한 에러 뿐만 아니라 then 메서드 내부에서 발생한 에러까지 모두 캐치할 수 있다.

```
promiseGet('대충 url')
    .then(res => console.xxx(res))
    .catch(err => console.error(err)); // TypeError: console.
```

- then 메서드에 두 번째 콜백 함수를 전달하는 것보다

  catch 메서드를 사용하는 것이 가독성이 좋고 명확하다.

45.5 프로미스 체이닝

- **프로미스 체이닝**

then, catch, finally 후속 처리 메서드는 언제나 프로미스를 반환하므로 연속적으로 호출 가능

후속 처리 메서드	콜백 함수의 인수	후속 처리 메서드의 반환값
then	promiseGet 함수가 반환한 프로미스가 resolve한 값(id가 1인 post)	콜백 함수가 반환한 프로미스
then	첫 번째 then 메서드가 반환한 프로미스가 resolve한 값(postId로 취득한 user 정보)	콜백 함수가 반환한 값(undefined)을 resolve한 프로미스

catch * 에러가 발생하지 않으면 호출되지 않는다.	promiseGet 함수 또는 앞선 후속 처리 메서드가 반환한 프로미스가 reject한 값	콜백 함수가 반환한 값(undefined)을 resolve한 프로미스

45.6 프로미스의 정적 메서드

45.6.1 Promise.resolve / Promise.reject

- 두 메서드는 이미 존재하는 값을 래핑하여 프로미스를 생성하기 위해 사용
- `Promise.resolve` 메서드는 인수로 전달받은 값을 `resolve`하는 프로미스를 생성
- `Promise.reject` 메서드는 인수로 전달받은 값을 `reject`하는 프로미스를 생성

45.6.2 Promise.all

- `Promise.all` 메서드는 여러 개의 비동기 처리를 모두 병렬 처리할 때 사용

```
// 세 개의 비동기 처리를 병렬로 처리
Promise.all([requestData1(), requestData2(), requestData3
()])
  .then(console.log) // [1, 2, 3] => 약 3초 소요
  .catch(console.error);
```

- `Promise.all` 메서드는 프로미스는 요소로 갖는 배열 등의 이터러블을 인수로 전달받는다.
- 전달받은 모든 프로미스가 모두 fulfilled 상태가 되면 모든 처리 결과를 배열에 저장해 새로운 프로미스를 반환한다.

45.6.3 Promise.race

- `Promise.race`는 Promise.all처럼 모든 프로미스가 fulfilled 상태가 되는 것을 기다리는 것이 아니라 **가장 먼저 fulfilled 상태가 된 프로미스의 처리 결과를 resolve**하는 새로운 프로미스를 반환

45.6.5 Promise.allSettled

- `Promise.allSettled` 메서드는 프로미스를 요소로 갖는 배열 등의 이터러블을 인수로 전달받는다.
- 전달받은 프로미스가 모두 settled 상태(비동기 처리가 수행된 상태,

즉 fulfilled 또는 rejected 상태)가 되면 처리 결과를 배열로 반환

45.7 마이크로태스크 큐

- 프로미스의 후속 처리 메서드의 콜백 함수는 태스크 큐가 아니라 마이크로태스크 큐에 저장
- **마이크로태스크 큐는 태스크큐보다 우선순위가 높다.**
  - 이벤트 루프는 콜 스택이 비면
    1. 먼저 마이크로태스크 큐에서 대기하고 있는 함수를 가져와 실행
    2. 이후 마이크로태스크 큐가 비면 태스크 큐에서 대기하고 있는 함수를 가져와 실행

45.8 fetch

- `fetch`는 XMLHttpRequest와 마찬가지로 HTTP 요청 전송 기능을 제공하는 클라이언트 사이드 Web API
- 프로미스를 지원하기 때문에 비동기 처리를 위한 콜백 패턴의 단점에서 자유롭다.
- **fetch 함수는 HTTP 응답을 나타내는 Response 객체를 래핑한 Promise 객체를 반환**

```
const request = {
    get(url) {
        return fetch(url);
    },
    post(url, payload) {
        return fetch(url, {
            method: 'POST',
            headers: { 'content-Type': : 'application/json'
},
            body: JSON.stringify(payload)
        });
    },
    patch(url, payload) {
        return fetch(url, {
            method: 'PATCH',
```

```
            headers: { 'content-Type': : 'application/json'
},
            body: JSON.stringify(payload)
        });
    },
    delete(url) {
        return fetch(url, {method: 'DELETE' });
    }
};
```